**Container with most water**

Se dă un șir de numere întregi (*height*) de lungime n. Există n linii verticale desenate astfel încât cele două capete ale liniei i sunt (i, 0) și (i, height[i]).

Găsește două linii care împreună cu axa x formează un container, astfel încât containerul să conțină cea mai multă apă.

Returnează cantitatea maximă de apă pe care un container o poate stoca.

* n == height.length
* 2 <= n <= 105
* 0 <= height[i] <= 104



**Input:** n = 9, height = [1,8,6,2,5,4,8,3,7]

**Output:** 49

**Constrangeri:**

n == height.length

2 <= n <= 105

0 <= height[i] <= 104

**Clase partitionare:**

1. C11: n < 2 si fiecare height se afla in [0, 10000]

int[] height = {}; // -1  
int[] height2 = {1000}; // -1

2. C12: n < 2 si cel putin un height nu se afla in [0, 10000]

int[] height = {-10}; // -1  
int[] height2 = {20000}; // -1

3. C21: n ∈ [2, 105] si fiecare height se afla in [0, 10000]

int[] height = {2, 4, 6, 3, 7, 10}; // 18  
int[] height2 = {100, 3, 400, 150, 200}; // 400

4. C22: n ∈ [2, 105] si cel putin un height nu se afla in [0, 10000]

int[] height = {2, -4, 6, 3, -7, 10}; // -2  
int[] height2 = {100, 3, 400, 150, 200000}; // -2

5. C31: n > 105 si fiecare height se afla in [0, 10000]

int[] arr = new int[120000];   
for(int i = 0; i < arr.length; i++){ arr[i] = i % 100;} // -1  
  
int[] arr2 = new int[110000];  
for(int i = 0; i < arr2.length; i++){ arr2[i] = i % 500 ;} // -1

6. C32: n > 105 si cel putin un height nu se afla in [0, 10000]

int[] arr = new int[120000];  
for(int i = 0; i < arr.length; i++){arr[i] = i % 100 - 2;} // -1  
  
int[] arr2 = new int[110000];  
for(int i = 0; i < arr2.length; i++){ arr2[i] = i % 500 - 4;} // -1

**Valori de frontiera:**

**C11**: [0] , [10000]

**C12**: [-1], [10001]

**C21**: [0, 0], [10000, 10000] || [0, 0, ...] , [10000, 10000, ...] de 100000 ori

**C22**: [-1, h], [10001, h] ||

[-1, -1] , [10001, 10001] ||

[-1, h1, h2 , ...] , [10001, h1, h2, ...] de 100000 ori ||

[-1, -1 ...] , [10001, 10001, ...] de 100000 ori

**C31**: [0, 0, 0, ..], [10000, 10000, ...] de 100001 ori

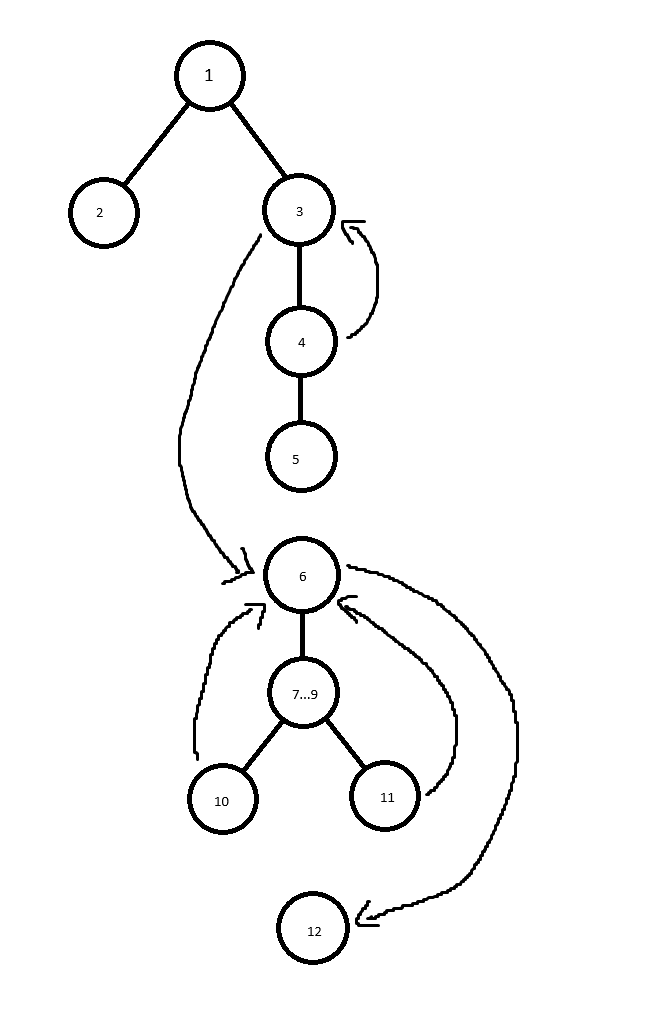
**C32**: [-1, h1, h2 , ...] , [10001, h1, h2, ...] de 100001 ori ||

[-1, -1 ...] , [10001, 10001, ...] de 100001 ori

public static int maxArea(int[] height, int n) {  
  
 int leftPointer = 0;  
 int rightPointer = n - 1;  
 int maxArea = 0;  
  
 if (n < 2 || n > 100000) { *//1* return -1; *//2* }  
  
 for (int i = 0; i < n; i++) { *//3* if (height[i] < 0 || height[i] > 10000) { *//4* return -2; *//5* }  
 }  
  
 while (leftPointer < rightPointer) { *//6* int currentArea = Math.*min*(height[leftPointer], height[rightPointer]) \* (rightPointer - leftPointer); *//7* maxArea = Math.*max*(maxArea, currentArea); *//8* if (height[leftPointer] < height[rightPointer]) { *//9* leftPointer++; *//10* } else {  
 rightPointer--; *//11* }  
 }  
 return maxArea; *//12*}

|  |  |
| --- | --- |
| 1 | if (n < 2 || n > 100000) { |
| 2 | return -1; |
| 3 | for (int i = 0; i < n; i++) { |
| 4 | if (height[i] < 0 || height[i] > 10000) { |
| 5 | return -2; |
| 6 | while (leftPointer < rightPointer) { |
| 7 | int currentArea = Math.*min*(height[leftPointer], height[rightPointer]) \* (rightPointer - leftPointer); |
| 8 | maxArea = Math.*max*(maxArea, currentArea |
| 9 | if (height[leftPointer] < height[rightPointer]) { |
| 10 | leftPointer++; |
| 11 | rightPointer--; |
| 12 | return maxArea; |

**Transformarea programului într-un graf orientat**



Pe baza grafului se pot defini diverse acoperiri:

* *Acoperire la nivel de instrucțiune*: fiecare instrucțiune (nod al grafului) este parcursă măcar o

dată

* *Acoperire la nivel de condiție*: fiecare condiție individuală dintr-o decizie să ia atât valoarea

adevărat cât și valoarea fals

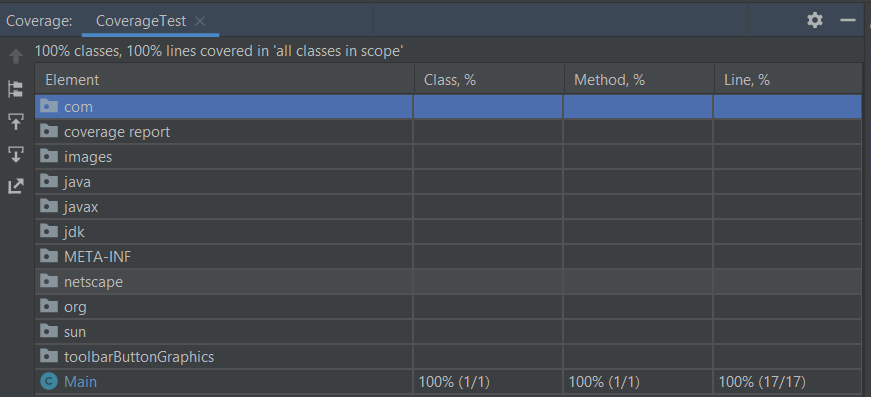
|  |  |  |
| --- | --- | --- |
|  | **Decizii** | **Conditii individuale** |
| 1 | if (n < 2 || n > 100000) | n < 2, n > 100000 |
| 2 | (int i = 0; i < n; i++) | i < n |
| 3 | (height[i] < 0 || height[i] > 10000) | height[i] < 0, height[i] > 10000 |
| 4 | while (leftPointer < rightPointer) | leftPointer < rightPointer |
| 5 | if (height[leftPointer] < height[rightPointer]) | height[leftPointer] < height[rightPointer]) |

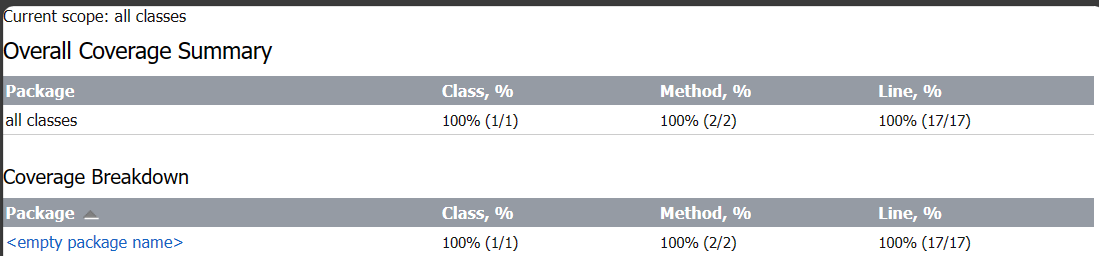
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Intrari** | | **Rezultat** | **Instructiuni parcurse** | **Decizii acoperite** |
| **n** | **heights** |
| 1 | {0} | -1 | 1, 2 | 1 |
| 4 | {3, 2, -1, 6} | -2 | 1, 3, 4, 3, 4, 3, 4, 5 | 1, 2, 3 |
| 4 | {3, 2, 7, 1} | 6 | 1, 3, 4, 3, 4, 3, 4, 3, 4, 3 6, 7…9, 11, 6, 7…9, 10, 6, 7…9, 10, 6, 12 | 1, 2, 3, 4, 5 |

|  |  |
| --- | --- |
|  | **Conditii individuale** |
| 1, 2 | n < 2, n > 100000 |
| 3 | i < n |
| 4, 5 | height[i] < 0, height[i] > 10000 |
| 6 | leftPointer < rightPointer |
| 7 | height[leftPointer] < height[rightPointer] |

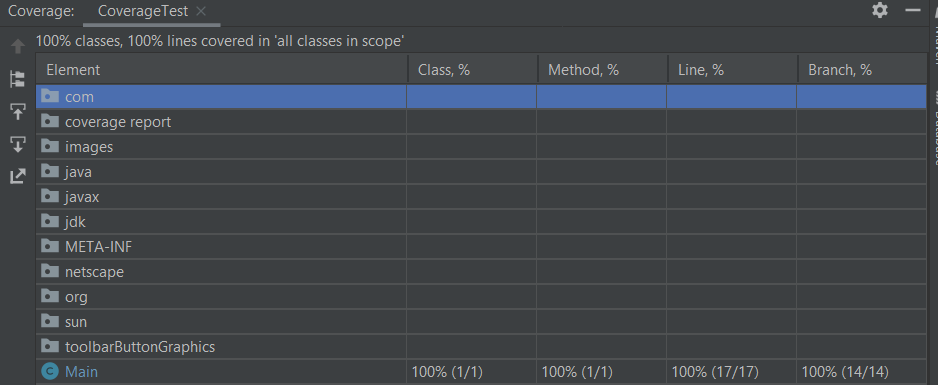
|  |  |  |  |
| --- | --- | --- | --- |
| **Intrari** | | **Rezultat** | **Conditii individuale parcurse** |
| **n** | **heights** |
| 1 | {0} | -1 | 1 |
| 100005 | {2, 2, …} | -1 | 2 |
| 4 | {3, 2, -1, 6} | -2 | 3, 4 |
| 3 | {10002, 4, 8} | -2 | 3, 5 |
| 4 | {3, 2, 7, 1} | 6 | 3, 6, 7 |

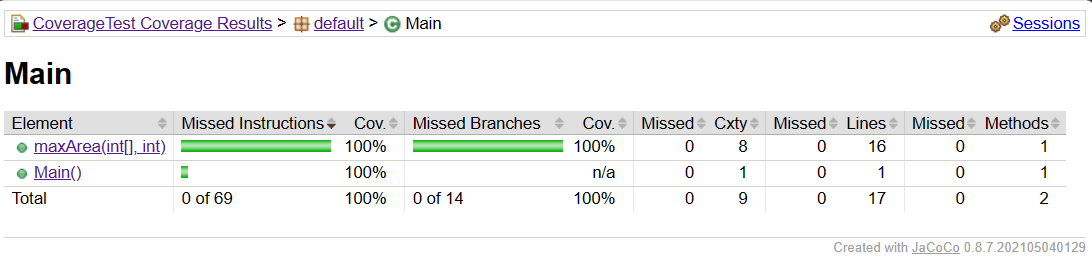
**Integrated Coverage Runner**

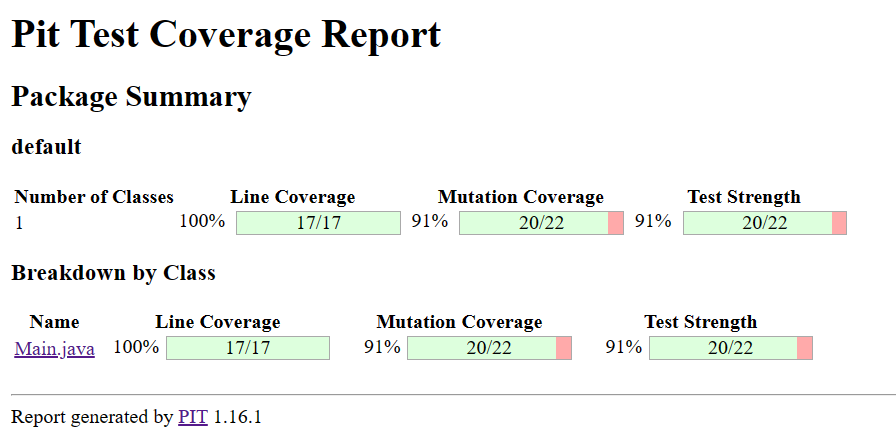


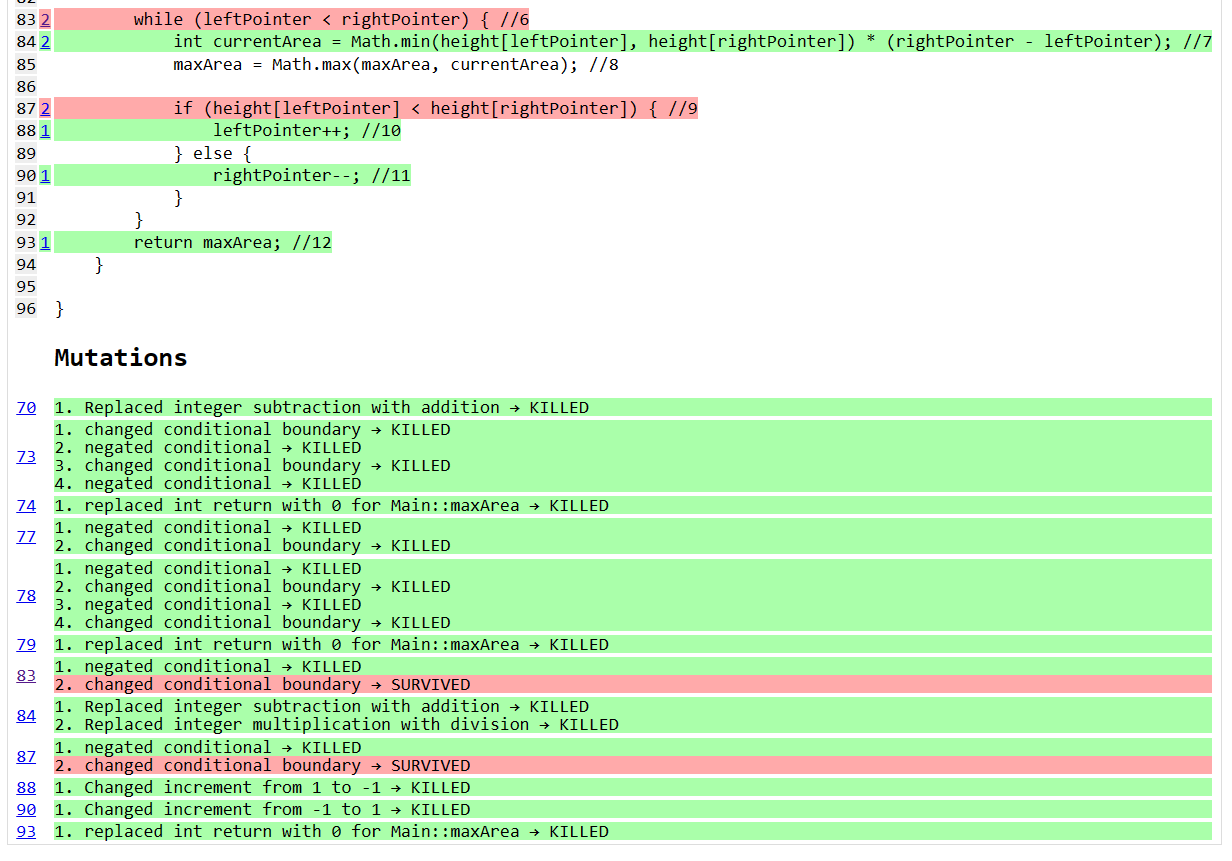


**JaCoCo Coverage Runner**









TODO: de adugat teste pentru a omori mutantii (modificarea facuta de mutant: ‘<=’)

**Referinte:** [Container With Most Water - LeetCode](https://leetcode.com/problems/container-with-most-water/description/)